Computer Science • 18(4) 2017 https://doi.org/10.7494/csci.2017.18.4.2361

Katarzyna Ścibisz-Mordelska
Radosław Nielek

LOWER PRECISION CALCULATION FOR OPTION PRICING

Abstract *The problem of option pricing is one of the most critical issues and fundamental building blocks in mathematical finance. The research includes the deployment of a lower-precision type in two option-pricing algorithms: the Black-Scholes and Monte Carlo simulations. We make an assumption that the shorter the number used for calculations is (in bits), the more operations we are able to perform at the same time. The results are examined by a comparison to the outputs of single- and double-precision types. The major goal of the study is to indicate whether the lower-precision types can be used in financial mathematics. The findings indicate that Black-Scholes provided more precise outputs than the basic implementation of the Monte Carlo simulation. Modification of the Monte Carlo algorithm is also proposed. The research shows the limitations and opportunities of the lower-precision type usage. In order to benefit from the application in terms of the time of calculation, the improved algorithms can be implemented on a GPU or FPGA. We conclude that, under particular restrictions, the lower-precision calculation can be used in mathematical finance.*

Keywords option pricing, lower precision, half-precision type, Monte Carlo, Black-Scholes

Citation Computer Science 18(4) 2017: 429–446

1. Introduction

An option is a derivative security that gives someone a right to buy (call option) or sell (put option) to its holder the underlying instrument for an agreed-upon price (strike price) on an agreed-upon date (expiry date). The most-popular underlying instruments are stocks, foreign exchanges, bonds, stock market indexes, or future contracts. Option pricing refers to the amount (premium) to be paid or received for an option contract. Option pricing is a demanding process due to its complexity. Depending on the type of option, there are a number of different variables to be taken into consideration. The two most-fundamental approaches are the Black-Scholes and Monte Carlo options model. The Black-Scholes formula was the first commonly used option-pricing model. It is applicable to European options and uses the current stock price, strike price, interest rates, time left to expiration, volatility, and dividend. In the Monte Carlo option-pricing model, the price is calculated as an option's discounted expected value. This method uses simulation to generate a large number of random price paths for the underlying instrument. The exercise value is calculated for each path, then the prices are averaged and discounted. New trading strategies like algorithmic trading require fast financial instrument calculation tools. Delivering valuations faster means a higher number of calculated financial instruments at the same time. It is a natural progress to adapt any existing algorithms and deliver new ones to accelerate valuations.

In order to accelerate calculations, researchers started to rebuild pricing algorithms to be performed in parallel. Many of them tried to deploy the algorithms on Graphical Processing Units (GPUs) as powerful, programmable, and cost-efficient computing architectures. Subsequent articles involve the use of Field-Programmable Gate Arrays (FPGA).

The main goal of the study is to reduce the precision of the calculations as compared to the precision used currently in the pricing process in such a way that it does not affect the final valuations. We make an assumption that the shorter the number used for calculations is (in bits), the more operations we are able to perform at the same time on a CPU as well as on a GPU. The calculations with the lower-precision type on a GPU is enabled by NVIDIA's CUDA by introducing new datatypes (half-precision type) and intrinsic functions for operating on them. An additional benefit of the use of the lower-precision type is a reduction in memory usage and smaller bandwidth requirements for the memory-processor bus.

In this work, we want to prove that the half-precision type can be successfully applied to option pricing with an acceptable aberration. Our contribution to achieve this goal encompasses:

- two modifications to the Monte Carlo algorithm,
- the confirmation that the Black-Scholes method can be applied without any alteration.

The results from the Black-Scholes approach are a suitable benchmark for further research. For the Monte Carlo method (due to the differences between architectures to be used for lower-precision type calculations), two modifications are proposed. For example, calculations denoted with HalfM1 are suitable for architectures that do not support the mixing of different precision types in calculations.

This paper is structured as follows: Section *Literature and related work* presents the latest implementations of option pricing using GPUs as a method of acceleration in the financial calculations. There is also a reference to the errors that are associated with the use of single- and double-precision types. This section provides examples of the implementations of the half-precision type (for example, in neural networks). Section *Research problem and thesis* outlines the main research questions and goals. Section *Analysis of the output* concerns the basic Monte Carlo simulation, Black-Scholes outputs, and two modifications of the Monte Carlo simulation. At the end of the study, we conclude and present possible future works.

2. Literature and related work

2.1. The performance of option pricing on GPU

A parallel implementation of binomial tree method on a GPU was described in [8]. Researchers studied the acceleration of binomial option pricing via parallelizing along time-axis on a GPU. The authors demonstrated the applicability of the approach to the pricing problem via binomial trees and its implementation on a GPU. In [14], the researchers discussed the problem of pricing Asian options with Black-Scholes and CUDA. The usage of the Quasi Monte Carlo simulation with a geometric Asian option as a control variate provided prices that are accurate with 2E-4 within a 50th of a second. The authors of [16] were interested in Monte Carlo simulations on a GPU and its optimizations. The proposed approach compacted the data to shrink the memory space with a crossing-path layout, which made GPU memory accesses possible. The developed Monte Carlo Exotic option-pricing approach was faster on the GPU than on the multi-core CPU.

The researchers in [6] obtained a speedup of about 18x for the double-precision calculations of a parallel implementation on a GPU (CUDA) of ADI time-discretization methods for European Rainbow and Basket options compared to the optimized sequential CPU computations. The equations were set in three spatial dimensions with mixed spatial derivatives in a variety of applications in computational finance. The results revealed the efficiency of the parallel methods. In [18], the authors reported an improvement in optimizing the Monte Carlo approach for the averages of 10 000 iterations, which overcame the limitations of the CPUs even for a moderate set of options. In [9], a significant speedup was achieved for the parallelization of the Black-Scholes, Monte Carlo, Bonds, and Repo code paths from the QuantLib library using hand-written CUDA and OpenCL codes that were specifically targeted for the GPU. In [7], the authors employed the Least Squared Monte Carlo

method on a GPU, which led to a reduction of computing time for the numerical pricing of European Multi-dimensional options based on the COS Method.

In [23], the researchers proved the acceleration of European and Bermudan option-pricing techniques based on Fourier cosine expansions. Then, [1] used GPU Monte Carlo European option pricing, which reduced the time of execution by 40× and energy consumption by 50×. The American options, which were implemented using the Longstaff and Schwartz regression method, were supplied with a speedup that varied between 2× and 10× according to the number of generated paths, dimensions, and time discretization. Authors of [22] explored the calculation of individual options with OpenCL and CUDA, proposing several parallel implementations of the Lattice model and Monte Carlo numerical pricing methods. The parallel implementations achieved a significant performance improvement over serial implementations.

The research conducted by [19] concerned the effects of errors in floating-point arithmetic in the context of the insufficiency of mathematical descriptions of models in the literature to replicate the outputs exactly in reference to real number arithmetic. The authors emphasized that, although the implemented code is correct, a computer platform used for computation can be responsible for *surprising* effects in its computations. The authors showed the impact of very simple modifications to the code (which should theoretically be mathematically equivalent). They considered FP32 and FP64 calculations. The authors pointed out interval arithmetic as a solution.

2.2. Use of FP16

Since the release of CUDA Toolkit 7.5 in the third quarter of 2015 [11], it is possible to launch efficiently native calculations on CUDA using FP16 as well as FP32 and FP64 types. This toolkit supports FP16 storage for up to 2× larger data sets and reduced memory bandwidth. The calculations with the FP16 type are enabled by adding new datatypes (*newhalf* and *half2*) and intrinsic functions for operating on them. The applications that previously suffered from the limited memory bandwidth may get up to a 2× speedup.

The half-precision floating-point type, according to the available literature, has gained popularity in neural networks. There is proof that the application of the half float-type may provide some benefits. The authors of [5] trained deep neural networks with low-precision storage and high-precision arithmetic on GPUs and FPGA. Researchers used a higher precision to store parameters, activations, and gradients. A lower precision was used for the forward and backward propagation. The results revealed that very-low-precision storage was sufficient not just for running trained networks but also for training them.

In [2], the authors used lower precision in the context of a parallel heterogeneous system, implementing a mixed-precision iterative refinement solver for substantial linear systems. Thanks to the combination of FPGA and GPU, it was possible to achieve a speedup of 3.7x for large dense $24\,064 \times 24\,064$ matrices compared to a tuned multi-threaded CPU solver based on the ATLAS linear algebra library. Further, [10]

studied the effect of limited-precision data representation and computation on neural network training. The results of their research revealed that the use of FP16 led to no degradation in the classification accuracy. According to [15] from the Intel company, there are many benefits resulting from the use of the half-precision type on a CPU too. As compared to the single-precision type, the half-precision type may fit into a lower level of cache with lower latency. Moreover, the reduced memory bandwidth may be freed up for other operations.

3. Research problem and thesis

The research concerns the problem of the FP16 type application in the valuation of financial options. This study is focused on the following questions: Is it possible to use the FP16 type in the valuation of financial options in order to obtain comparable results to those obtained using the FP32 and FP64 types? If the precision is satisfying, which algorithms are possible to be applied? What are the sources of errors? The following research thesis appears: In spite of the lower precision of the half-precision type, it is possible to obtain valuations of European options that fit within an acceptable aberration in comparison to the pricing with a single or double precision.

The research is based on the available literature concerning the improvement of the option pricing using a CPU and GPU. To the best of our knowledge, there has been no research focusing on the implementation of the half-precision type on a GPU for option pricing. The most-representative and most-popular methods of option valuation are the Black-Scholes formula and Monte Carlo simulation. It is a proper base for applications of any further pricing algorithms.

4. Option pricing

4.1. Option pricing – basic concepts

An option is one of the most-fundamental terms in mathematical finance. An option is a derivative security that gives someone a right to buy (call options) or sell (put option) the underlying instrument for an agreed-upon price (strike price, exercise price) on an agreed-upon date [3]. The most-popular underlying instruments are stocks, foreign exchanges, bonds, stock market indexes, or future contracts. Each financial option has its own specification. The strike price (exercise price) is an agreed-upon price for an underlying asset at which exercise will occur. The expiration date is the last date an option can be exercised. Each options also has settlement terms that indicate when a contract can be exercised, for example. In terms of the expiry date, different styles of options may be distinguished.

The most-basic ones are European and American options [21]. These are called vanilla options. American options can be exercised on any trading day on or before it expires. Contrary to the American, European options can only be exercised on

the expiration date. Exotic options is a wide category including options with various types of exercise or calculation of a payoff. An Asian option payoff is dependent on the average value of an underlying instrument until it expires. The average may be calculated as an arithmetic or geometric mean. Barrier option exercise depends on reaching an agreed-upon barrier level by the price of the underlying instrument. For a basket option, the underlying asset constitutes a group of commodities, securities or currencies. Rainbow options are vulnerable to two or more sources of uncertainty, contrary to simple types of options that are exposed to one source (the price of an underlying asset).

Option pricing refers to the amount to be paid or received for an option contract. Factors that affect the value of an option include the current price of a stock (spot price), time left to expiration, volatility, interest rates, and dividend [12]. Option pricing is a demanding process due to its complexity. Depending on the type of option, there are a number of manifold variables to be taken into consideration. The most-common models are the Black-Scholes model, binomial option-pricing model, trinomial trees approach, Monte Carlo option model, greeks, finite difference methods for options, and many others. These models are constantly being developed. Computational finance is a field of applied computer science that exploits advanced computation methods to deliver efficient financial models or systems [17]. Computational Finance has three major applications: Algorithmic Trading (AT), High-Frequency Trading (HFT), and Mathematical finance. The new trading strategies require fast financial instrument calculation tools. Delivering valuations faster means a higher number of calculated financial instruments at the same time. It is a natural progress to adapt any existing algorithms and deliver new ones to accelerate valuations.

We decided to choose two basic algorithms that are still used by many financial institutions. These models are the Black-Scholes and Monte Carlo simulations, which constitute a suitable benchmark for any further consideration. Black-Scholes involves basic calculations, while Monte Carlo deploys simple calculations with simulation. Considering the vanilla options, the Black-Scholes model is only applicable for options that can be exercised at one point in time. These are European options.

4.2. Black-Scholes

An option is one of the derivative securities. Its price is dependent on the price or value of the basic security. The basic security is called an underlying security [3]. An option's holder has a right to buy (sell) the call (put) option at exercise price (K). Contrary to the American option, European options can be exercised only at maturity (T). A call option exercised at maturity gives payoff $C_T = max(0, S_T - K)$, and a payoff for a put option equals $P_T = max(0, K - S_T)$. S_t is the price or value of the underlying security at time t. A call and put option priced for date t equal (respectively):

$$c = SN(d_1) - Ke^{-r(T-t)}N(d_2) \tag{1}$$

$$p = Ke^{-r(T-t)}N(-d_2) - SN(-d_1) \tag{2}$$

where:

$$d_1 = \frac{1}{\sigma\sqrt{T-t}}(\ln(\frac{S}{K}) + (r + \frac{1}{2}\sigma^2)(T-t)) \tag{3}$$

$$d_2 = d_1 - \sigma\sqrt{T-t} \tag{4}$$

According to the assumptions on the assets, r is the continuously compounded risk-free interest rate. Volatility is represented by σ. This is the standard deviation of the underlying asset. $N(\cdot)$ is the cumulative normal distribution.

4.3. Monte Carlo

The underlying method used in the Monte Carlo experiment is the Black-Scholes equation [4]. As previously outlined, a call option price equals $C_T = max(0; S_T - K)$ at maturity. In the Monte Carlo simulation, an option's price at date t equals its expected present value:

$$C_t = E[PV(max(0; S_T - K))] \tag{5}$$

Adding the assumption of *risk neutral result*, C_t can be calculated as:

$$C_t = e^{-r(T-t)}E^*[max(0; S_T - K)] \tag{6}$$

where E^* indicates a transformation of the former expected value. In order to estimate a call price, there is a need to conduct simulations of a large number of sample values of S_T. The estimated price is an arithmetic average of the simulated values.

Regarding the law of large numbers, the average should converge to the actual call price. The deployed number of simulations has an influence on the rate of convergence. The calculation of a call price by simulation involves simulating the terminal price of the underlying security, which equals S_T. The interval between t and T is irrelevant; therefore, the average of option payoffs at maturity is estimated: $E^*[max(0; S_T - K)]$. Additionally, it has to be discounted at the risk free rate:

$$c_t = e^{-r(T-t)}\frac{1}{n}\sum_{i=1}^{n} max(0; S_{T,i} - K)$$

where n is the number of simulations. Respectively, the put price equals:

$$p_t = e^{-r(T-t)}\frac{1}{n}\sum_{i=1}^{n} max(0; K - S_{T,i}).$$

5. Analysis of output

There were three data representation types used in the study: half-, single-, and double-precision floating-point. According to IEEE Standard 754 [13], the half-type consists of 1 bit sign, 5 bits exponent, 10 bits mantissa, and this type uses all of the standard IEEE rules like infinities, NaNs, and denormals. The range of the half-type

is limited. The representable minimum and maximum is (respectively): $5.96 \cdot 10^{-8}$ and 65 504. The half-precision library in version 1.11.1 used in the calculations provides an IEEE Standard 754 16-bit half-precision floating-point type. Arithmetic operators, type conversions, and commonly used mathematical functions are also available [20].

The Black-Scholes approach requires initial values of the following parameters:

- S – current value of the underlying asset,
- K – strike price,
- r – risk-free interest rate,
- σ – volatility,
- *time* – time left to maturity.

Additionally, the Monte Carlo approach demands a number of simulations (*nsims*). Each simulation was repeated 100 and 500 times (*nreps*) in order to collect basic statistics like mean, median, and minimum and maximum prices. Finally, 2 800 pairs of Black-Scholes put and call mean valuations and 28 000 pairs of mean put and call Monte Carlo valuations were produced.

The initial task of the research was focused on replacing the former representation types of the variables with the half-type in the Monte Carlo simulation (section 5.1) and Black-Scholes approach (section 5.2). The algorithms were not modified except for changing the variable representation types (for example, from double to half) to limit the precision.

Unfortunately, the results of achieved in the basic application of the Monte Carlo approach were not accurate. The first modification (Section 5.3) introduced an auxiliary variable represented by a half-precision type. This variable was responsible for collecting the partial sums in order to improve the accuracy of the outputs. The outputs provided by this method were denoted with HalfM1. The second modification – HalfM2 (Section 5.4) – regards a situation when the architecture of the system supports the use of mixed-data types. The variable that gathered the sum of the payoffs was represented with a single-precision type. It acted as an accumulator in the simulation.

5.1. Monte Carlo simulation

Performed valuations reveal that, in the Monte Carlo method, numerous missing put and call price pairs (10 981 out of 28 000) were produced, where at least one of them indicated infinity. The number of invalid pairs of prices grows with the number of simulations. For example, for nsims = 1000, the quantity equals 1791, and for nsims = 5000, this reaches 2463.

The problem stems from the accumulation of the sum of the payoffs in the Monte Carlo simulation. The total sum of payoffs is divided by the number of simulations. There is a real possibility of exceeding the range in calculations with the half-precision format. For example, this occurs for a call valuation for the following set of the parameters: S = 78.89, K = 78.89, r = 0.5, $\sigma = 0.7$ and time = 0.3. The sum reaches the value of 65 504 and then becomes infinite. It occurs for the 4684^{th} simulation

step. Further analysis reveals that, with the growth of the number of simulations, the precision drops. These two effects can be illustrated in Figure 1; here are presented half (gray dashed line), float (gray dotted line) and double (solid gray line) call prices for 5000 simulations. Since the 500^{th} step of a simulation, half valuation gradually begins to fall while the double and float prices converge. Consistently, prices for the 4684^{th} simulation step and further are not available.

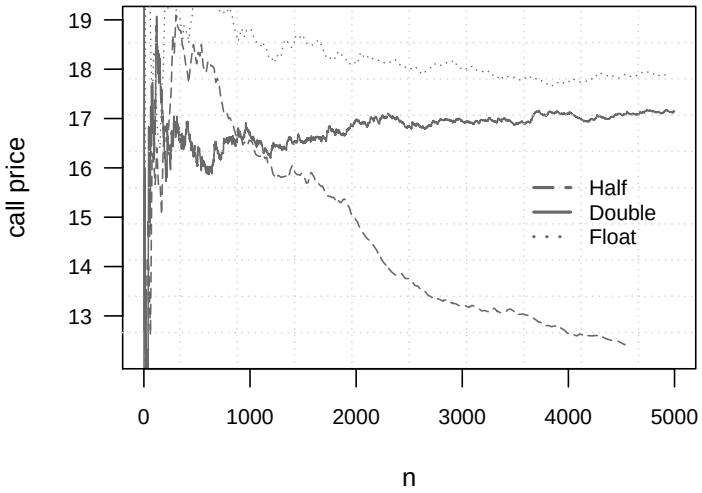

Figure 1. Half, Float, Double Call prices for S = 78.89, K = 78.89, r = 0.5, $\sigma = 0.7$, time = 0.3, and nrep=1.

5.2. Black-Scholes output

The outputs of the Black-Scholes calculations are in some ways more rewarding. No infinite values were produced. Moreover, half outputs correspond with those provided by the float or double. The sample results in Table 1 present the overall performance of the calculations.

Table 1
Black-Scholes put and call prices for S = 100.991, K = 140, r = 0.5, and $\sigma = 0.2$.

Time	Price	Half	Float	Double
0.3	put	11.1719	11.1786	11.1786
0.3	call	0.0541077	0.0587816	0.058782
0.6	put	10.8281	10.8381	10.8381
0.6	call	0.338867	0.349241	0.34924

The differences between the float and double prices basically do not exist. The differences between the prices calculated with the half-type and float or double are fractional. However, these fractional disparities are the most problematic to the prices

below unity, especially those striving to zero. Table 2 provides examples of negative prices that are instances of the problems with rounding the fractional parts of the numbers. The first visible cause is the set of initial parameters and the way the half format represents them (Tab. 3).

Table 2
Black-Scholes prices for S = 100.991, K = 140, r = 0.001.

Time	σ	Price	Half	Float	Double
0.3	0.2	put	39	38.9724	38.9724
0.3	0.2	call	−0.00786591	0.00535116	0.00535322
0.6	0.1	put	38.9375	38.9251	38.9251
0.6	0.1	call	−0.0190277	0.0000315246	0.0000254553

Table 3
Examples of parameters in half arithmetic.

Variable	Value	Half representation
S	100.991	100.938
r	0.001	0.000999451
σ	0.2	0.199951
time	0.6	0.599609

5.3. Half pricing by simulation – modification

The first modification, denoted as HalfM1, regards the introduction of an auxiliary variable in the Monte Carlo simulation. It was an experimental remedy for the declining precision of the results with the increasing number of simulations. This idea was applied bearing in mind the situations when, due to the architecture of a system, it is not possible to mix various data types. It has its limitations, since it does not eliminate the problem with infinite values.

In the basic Monte Carlo approach, partial sums were added to the sum of the payoffs in each run of the loop. The application of the half-type in the algorithm caused situations in which some of the partial sums were too modest to have an impact on the total sum of the payoffs. The modification involved adding an auxiliary variable that collected the partial sums until the auxiliary variable reached an arbitrarily chosen value. This value was the last positive integer value that can be exactly represented with a half-type – this is 2048.

After reaching this value, the amount accumulated by the auxiliary variable is added to the sum of the payoffs, the auxiliary variable becomes zero, and the process is repeated. Unfortunately, the introduced modification does not lead to a decrease in the number of missing values that are the cause of exceeding the range. Eventually, the sum of the payoffs calculated with the modification will not be lower than the sum

calculated with the primary solution. This can result in a situation when previously underestimated valuations exceed the range.

The problem still happens to set S = 78.89, K = 78.89, r = 0.5, time = 0.3, $\sigma = 0.7$ and nrep = 1 for call price (Fig. 2) where, with an increasing number of simulations, the pricing becomes infinity and is not available (HalfM1, black dotted line). A higher number of simulations means either a growing or constant sum of payoffs. The accumulating partial sums are captured; therefore, the valuation is more-precise. The limit is reached for a smaller number of simulations than for the basic Monte Carlo pricing function. Nonetheless, the prices are converging now. The put price for the same set of initial parameters also performed better (Fig. 3). One can see that the new method of pricing indicates improvement in the pricing process (HalfM1).

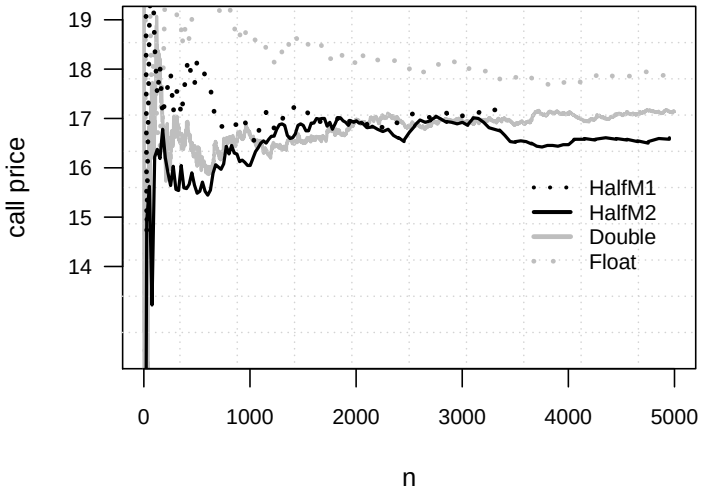

Figure 2. HalfM1, HalfM2, Float, Double Call prices for S = 78.89, K = 78.89, r = 0.5, $\sigma = 0.7$, time = 0.3, and nrep = 1.

This modification was an experimental way of improving the pricing process. In a certain way, it delivers more-precise valuations; for example, for set S = 78.89, K = 78.89, r = 0.1, time = 1, $\sigma = 0.1$, nsims = 5000, and nrep = 100 for call prices. After the deployment of a new formula, there was a jump from 4.38938 to 7.84004 (Tab. 4), which is a considerable outcome. The distance between the new half prices and float or double prices is still significant as compared with the distance between the float and double prices. The valuations below unity are not as properly performed as those above unity. The mean put half price obtained by the pricing function after modification for S = 78.89, K = 78.89, r = 0.5, time = 0.02, $\sigma = 0.2$, nsims = 2000, and nrep = 100 (Tab. 4) is still closer to the basic half price. It appears that the modification does not grant an improvement for prices under unity. Notwithstanding, the call price experienced a recovery from 2.07435 to 2.36568.

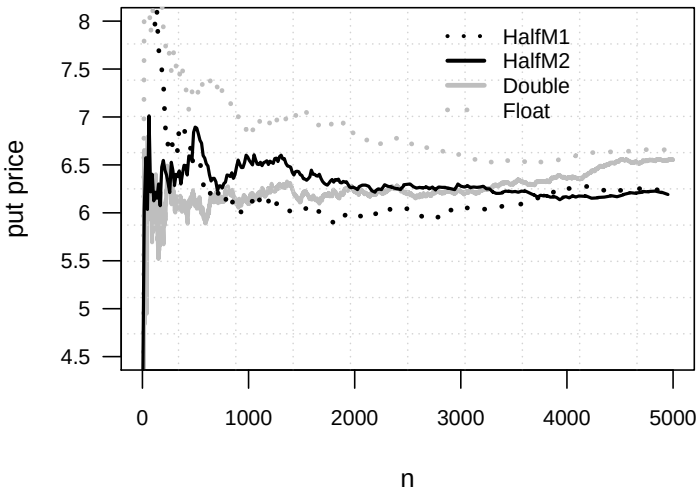

Figure 3. HalfM1, HalfM2, Float, Double Put prices for S = 78.89, K = 78.89, r = 0.5, $\sigma = 0.7$, time = 0.3, and nrep = 1.

Table 4
Half, HalfM1, Float and Double call and put prices: S = 78.89, K = 78.89, and nrep = 100.

r	Time	σ	nsim	Type	Put	Call
0.1	1	0.1	5000	HalfM1	0.58273	7.84004
0.1	1	0.1	5000	Half	0.540288	4.38938
0.1	1	0.1	5000	Double	0.626697	8.13098
0.1	1	0.1	5000	Float	0.624729	8.13021
0.5	0.02	0.2	2000	HalfM1	0.564639	2.36568
0.5	0.02	0.2	2000	Half	0.567339	2.07435
0.5	0.02	0.2	2000	Double	0.634074	2.57531
0.5	0.02	0.2	2000	Float	0.624683	2.57529

The introduced modification for set S = 10.23, K = 22, r = 0.1, $\sigma = 0.7$, time = 0.3, and nsims = 5000 provides sufficient results (Fig. 4). The new half call price (HalfM1 – black dotted line) coincides with the double and float prices, and the double call price is accompanied by the half price for the majority of the simulations. It is hard to clearly say that the modification brought exact valuations. A brief analysis of the boxplots for the same set with nrep = 500 for the put price after modification (Fig. 5) indicates that the mean put price has moved closer to the double and float prices. One can see that the modification brought an improvement regarding some sets of the initial parameters. However, there are two main drawbacks for this method. The first includes a significant number of unavailable prices due to the surpassed range. After modification, there were about 880 more infinity values than before the addition of

the auxiliary variable. The second disadvantage is the poor condition of the prices for some specific sets of variables.

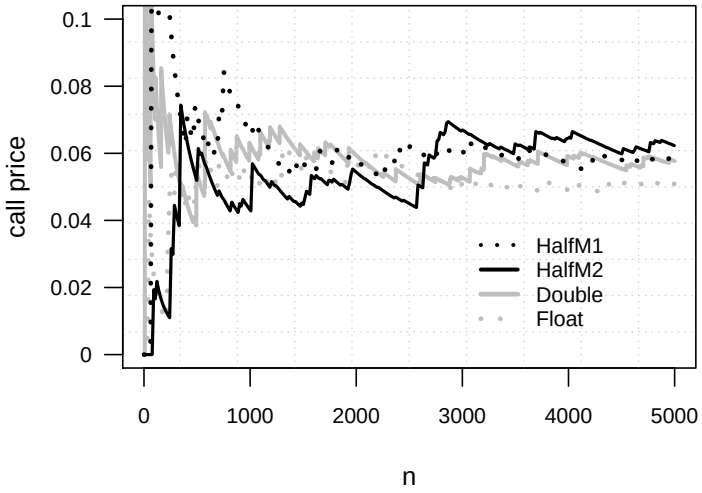

Figure 4. HalfM1, HalfM2, Float, Double Call prices for S = 10.23, K = 22, r = 0.1, $\sigma = 0.7$, time = 0.3, and nrep = 1.

5.4. Half pricing by simulation with float-type

The results obtained by the presented methods are not sufficient to perform a reliable output. A second modification of the basic half-pricing function was proposed in which the sum of the payoffs was represented by a single floating-point type. This action prevents the sum of the payoffs from exceeding the range, and it solves two main problems: the presence of infinite values and the lack of precision. For the whole initial set of parameters, it was possible to obtain all of the valuations. The second modification, denoted as HalfM2, involves the use of a float-type variable for the accumulation of the sum of the payoffs.

This modification regards the situation when the architecture of the system supports a use of mixed data types. This single-precision type variable worked as an accumulator in the simulation. As a result, it was possible to obtain all of the valuations with improved precision. The sum of the payoffs for the call prices for set S = 78.89, K = 78.89, r = 0.5, $\sigma = 0.7$, time = 0.3, and nrep = 1 indeed reaches a value of 65 504 for the 3268^{th} step of the simulation (like in the case with an auxiliary variable) and then continues collecting the summands (Fig. 2). It is clear now that the call-half valuation (HalfM2 – black solid line) for the following number of simulations is reachable and follows the double price path. The outputs for the put valuations are quite satisfying as well (Fig. 3). The mean values for the call and put prices for nrep = 500 (Tab. 5) resolves any doubts. The mean values of HalfM2 provides sufficient outputs.

The call valuation for S = 10.23, K = 22, r = 0.1, $\sigma = 0.7$, time = 0.3, nsims = 5000, and nrep = 1 is acceptable as well (Fig. 4). What is worth noticing, all mean call prices for this set of initial values that are mentioned in Table 5 perform fairly well. Step by step, the put prices climb and provide a comparable output: 11.1826 for HalfM2 and 11.1808, 11.1785 for the double and float prices.

Table 5

Half, HalfM1, HalfM2, Float Double call, and put prices for nsims = 5000 and nrep = 500.

S	K	r	Time	σ	Type	Put	Call
78.89	78.89	0.5	0.3	0.7	HalfM2	6.49563	17.4358
78.89	78.89	0.5	0.3	0.7	HalfM1	6.34454	inf
78.89	78.89	0.5	0.3	0.7	Half	4.69036	inf
78.89	78.89	0.5	0.3	0.7	Double	6.49423	17.4669
78.89	78.89	0.5	0.3	0.7	Float	6.49674	17.5025
10.23	22	0.1	0.3	0.7	HalfM2	11.1826	0.0587925
10.23	22	0.1	0.3	0.7	HalfM1	10.8114	0.0575419
10.23	22	0.1	0.3	0.7	Half	4.13828	0.0579714
10.23	22	0.1	0.3	0.7	Double	11.1808	0.0588854
10.23	22	0.1	0.3	0.7	Float	11.1785	0.0584732

A brief analysis of the boxplots for set S = 10.23 K = 22, r = 0.1, r = 0.7, time = 0.3, nsims = 5000, and nrep = 500 for the put price after modification with the float variable (Fig. 5) indicates that the mean prices are nearly the same despite what was illustrated by the charts of valuations at each step of the simulation. The results obtained by this form of modification of the Monte Carlo formula are more rewarding in most cases. The precision of the valuations for both the call and put prices is much higher. Better results are obtained even for those valuations below unity.

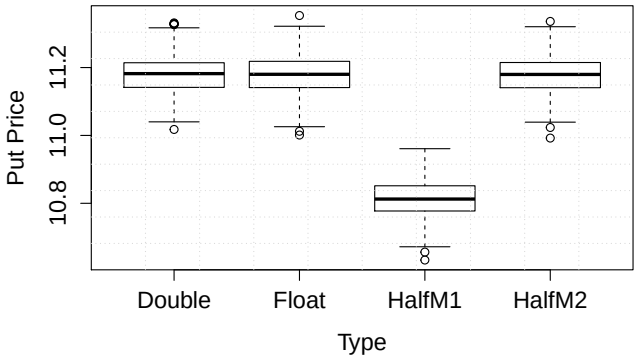

Figure 5. Boxplots. HalfM1, HalfM2, Float, Double Put prices for S = 10.23, K = 22, r = 0.1, $\sigma = 0.7$, time = 0.3, nsims = 5000 and nrep = 500.

6. Conclusions and future works

6.1. Results

We were focused on the use of basic unaltered forms of the Black-Scholes and Monte Carlo algorithms. The first use of the Monte Carlo algorithm with the FP16 type was not sufficient to supply satisfying results. When the outputs were available, many of them suffered from the lack of precision. By contrast, all of the Black-Scholes valuations were produced with an acceptable precision. They outperformed those delivered by the basic Monte Carlo simulation. The first modification of the Monte Carlo simulation provided a lower number of non-missing valuations. Nevertheless, the results of the first modification were double-sided. The precision of the options was enhanced; however, this caused the unavailability of many of them. The partial sums were collected more precisely, which resulted in the surpassing of the range of the half-precision type. In the second modification, all of the valuations were available, and the precision was improved. Nevertheless, this approach has its limitations considering the architecture to be used for calculations. Overall conclusions may indicate that there is a need to analyze a number of repetitions of the pricing algorithms that retrieve the basic statistics. An analysis of a single run of the Monte Carlo simulation is also necessary for an examination of the road where the simulation goes.

Under particular conditions, it is possible to use the FP16 type in option pricing obtaining reasonable outputs. Black-Scholes as well as the mixed-precision Monte Carlo solution provide the best results. The limitations of the FP16 type are directly linked to its range for particular sets of parameter values. The distorted values of the initial parameters represented by FP16 as well as the problems with rounding under unity and over 65 504 are the main sources of errors in the calculations.

6.2. The contribution

The problem of option pricing is one of the most-critical issues and fundamental building blocks in mathematical finance. This means that, in the most cases, the process must be performed in real time. Financial computations with the FP16 type not only offer new possibilities but also bring new challenges.

The conducted research is important for a number of reasons. The presented related work focused mainly on the application of single- and double-precision types, and none of the studies showed a deployment of a lower-precision type in terms of option pricing. In our study, we show the outputs of the basic implementations of FP16 on the proposed algorithms and explain the limitations of the FP16 type in the presented methods. We have shown that the Black-Scholes algorithms with the FP16 type supply acceptable outputs without any alterations. These results are a sufficient benchmark for further calculations. We have also identified the main problems resulting from the shift from single- or double- to lower-precision types in the option-pricing algorithms. The most visible and influential difficulties were encountered in the Monte Carlo approach. We have developed two useful modifications to enhance the compu-

tations due to the differences between the architectures offering lower-precision types. Calculations denoted with HalfM1 are suitable for architectures that do not support mixing different precision types in their calculations. The modification named HalfM2 is applicable for more-flexible architectures. Both modifications may be useful for any other research than option pricing.

The significance of this work is attached to the use of a new native precision type in pricing algorithms. To the best of our knowledge, none of the existing research has introduced the lower-precision type in terms of option pricing. Most of the presented studies focused on accelerating the calculations. By contrast, we tried to provide accurate results. Acceleration is possible if the results calculated with the low-precision type are accurate. Acceleration may require usage of, e.g. a GPU and rewriting the same algorithms in the CUDA framework using lower precision.

6.3. Future works

Having algorithms that provide accurate results, it is justified to implement them on various architectures (including GPUs) in order to check the performance, effectiveness, and precision of the calculations. The stress should be put on the benefits of the use of a half-precision type on a GPU. Further research may involve hermetic modifications and an exact evaluation of the inaccuracy sources. The implementations should also be verified on real examples. One way of developing this study includes the construction of a classifier or a collection of heuristics, which will decide which type of precision is suitable for a given algorithm.

References

[1] Abbas-Turki L.A., Vialle S., Lapeyre B., Mercier P.: Pricing derivatives on graphics processing units using Monte Carlo simulation, *Concurrency and Computation: Practice and Experience*, vol. 26(9), pp. 1679–1697, 2014.

[2] Angerer C.M., Polig R., Zegarac D., Giefers H., Hagleitner C., Bekas C., Curio-ni A.: A fast, hybrid, power-efficient high-precision solver for large linear systems based on low-precision hardware, *Sustainable Computing: Informatics and Systems*, vol. 12, pp. 72–82, 2015. http://dx.doi.org/10.1016/j.suscom.2015.10.001.

[3] Black F., Scholes M.: The Pricing of Options and Corporate Liabilities, *Journal of Political Economy*, vol. 81(3), pp. 637–654, 1973. http://EconPapers.repec.org/RePEc:ucp:jpolec:v:81:y:1973:i:3:p:637-54.

[4] Boyle P.P.: Options: A Monte Carlo approach, *Journal of Financial Economics*, vol. 4(3), pp. 323–338, 1977. https://ideas.repec.org/a/eee/jfinec/v4y1977i3p323-338.html.

[5] Courbariaux M., Bengio Y., David J.: Training deep neural networks with low precision multiplications. In: *CoRR*, vol. abs/1412.7024, 2014. http://arxiv.org/abs/1412.7024.

[6] Dang D.M., Christara C.C., Jackson K.R.: A parallel implementation on GPUs of ADI finite difference methods for parabolic PDEs with applications in finance, *Canadian Applied Mathematics Quarterly*, vol. 75(4), pp. 627–660, 2009. http://ami.math.ualberta.ca/CAMQ/pdf_files/vol_17/17_4/17_4b.pdf.

[7] Fatica M., Phillips E.: Pricing American options with least squares Monte Carlo on GPUs. In: *Proceedings of the 6th Workshop on High Performance Computational Finance*, p. 5. ACM, 2013.

[8] Ganesan N., Chamberlain R.D., Buhler J.: Acceleration of Binomial Options Pricing via Parallelizing along Time-axis on a GPU. In: *Proceedings of Symposium on Application Accelerators in High Performance Computing*. 2009.

[9] Grauer-Gray S., Killian W., Searles R., Cavazos J.: Accelerating financial applications on the GPU. In: *Proceedings of the 6th Workshop on General Purpose Processor Using Graphics Processing Units*, pp. 127–136. ACM, 2013.

[10] Gupta S., Agrawal A., Gopalakrishnan K., Narayanan P.: Deep Learning with Limited Numerical Precision. In: *CoRR*, vol. abs/1502.02551, 2015. http://arxiv.org/abs/1502.02551.

[11] Harris M.: New Features in CUDA 7.5, 2015. http://devblogs.nvidia.com/parallelforall/new-features-cuda-7-5/.

[12] Haug E.G.: *The complete guide to option pricing formulas*, McGraw-Hill Companies, 2007.

[13] IEEE Task P754: *IEEE 754-2008, Standard for Floating-Point Arithmetic*, 2008. http://dx.doi.org/10.1109/IEEESTD.2008.4610935.

[14] Joshi M.S.: Graphical Asian options, *Wilmott Journal*, vol. 2(2), pp. 97–107, 2010. http://dx.doi.org/10.1002/wilj.26.

[15] Konsor P.: Performance Benefits of Half Precision Floats, Intel Developer Zone, 2012. https://software.intel.com/en-us/articles/performance-benefits-of-half-precision-floats.

[16] Liu L., Zhang Y., Yang G., Zheng W.: Efficient Monte Carlo-based options pricing on graphics processors and its optimizations, *Science China Information Sciences*, vol. 53(9), pp. 1703–1712, 2010. http://dx.doi.org/10.1007/s11432-010-3109-7.

[17] Los C.: *Computational Finance: A Scientific Perspective*. World Scientific Pub., 2001. https://books.google.ca/books?id=dLWmQgAACAAJ.

[18] Lungu I., Petroşanu D.M., Pîrjan A.: Solutions for optimizing the Monte Carlo option pricing method's implementation using the compute unified device architecture, *University Politehnica of Bucharest Scientific Bulletin, Series A: Applied Mathematics and Physics*, vol. 75(3), pp. 105–112, 2013.

[19] Polhill J.G., Izquierdo L.R., Gotts N.M.: The Ghost in the Model (and Other Effects of Floating Point Arithmetic), *Journal of Artificial Societies and Social Simulation*, vol. 8(1), pp. 1–5, 2004.

[20] Rau C.: Half: Half-precision floating point library, 2013. `http://half.sourceforge.net/`.

[21] Sharpe W.F., Alexander G.J., Bailey J.V.: *Investments*, vol. 6. Prentice-Hall Upper Saddle River, NJ, 1999.

[22] Suo S., Zhu R., Attridge R., Wan J.: GPU option pricing. In: *Proceedings of the 8th Workshop on High Performance Computational Finance*, WHPCF '15, pp. 8:1–8:6. ACM, New York, NY, USA, 2015. `http://dx.doi.org/10.1145/2830556.2830564`.

[23] Zhang B., Oosterlee C.W.: Acceleration of option pricing technique on graphics processing units, *Concurrency and Computation. Practice and Experience*, vol. 26(9), pp. 1626–1639, 2014. `http://dx.doi.org/10.1002/cpe.2825`.

Affiliations

Katarzyna Ścibisz-Mordelska
Polish-Japanese Academy of Information Technology, Koszykowa 86, 02–008 Warszawa, Poland, `katarzyna.scibisz@pja.edu.pl`

Radosław Nielek
Polish-Japanese Academy of Information Technology, Koszykowa 86, 02–008 Warszawa, Poland, `nielek@pja.edu.pl`

Received: 08.02.2017
Revised: 03.09.2017
Accepted: 14.09.2017